Available online at www.sciencedirect.com

ScienceDirect

Artificial Intelligence 171 (2007) 434–439

Artificial Intelligence

www.elsevier.com/locate/artint

# No regrets about no-regret

Yu-Han Chang

*Intelligent Systems Division, USC Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, USA*

Received 16 May 2006; received in revised form 26 October 2006; accepted 13 December 2006

Available online 13 February 2007

**Abstract**

No-regret is described as one framework that game theorists and computer scientists have converged upon for designing and evaluating multi-agent learning algorithms. However, Shoham, Powers, and Grenager also point out that the framework has serious deficiencies, such as behaving sub-optimally against certain reactive opponents. But all is not lost. With some simple modifications, regret-minimizing algorithms can perform in many of the ways we wish multi-agent learning algorithms to perform, providing safety and adaptability against reactive opponents. We argue that the research community should have no regrets about no-regret methods.



*Keywords:* Multi-agent learning; Regret-minimization; Game theory

## 1. Introduction

Traditional no-regret algorithms sometimes perform sub-optimally because the regret criterion only compares the algorithm's performance to the possible alternate outcomes in the individual stage games of a repeated game, assuming the opponent's action stays fixed. For example, a typical no-regret agent playing Prisoner's Dilemma would end up always defecting, since this minimizes the algorithm's regret relative to the other possible action of cooperating, given the observed sequence of opponent actions. However, it is relatively simple to extend standard no-regret approaches to handle and avoid these suboptimal cases. In fact, I would argue that such modified no-regret algorithms hold much promise for the future direction of multi-agent learning.

Regret minimization methods, sometimes also referred to as experts algorithms or hedging algorithms, provide the clearest method for evaluating agent performance in general multi-agent settings. Since we would often like to assume that the opponent in multi-agent learning problems is unknown, it is usually difficult to evaluate agent performance, which depends on the type of opponent the agent ends up playing against. Regret-minimization approaches circumvent this problem by defining performance in terms of a comparison class of possible strategies that the agent itself is capable of executing. Thus, no assumptions need to be made about the opponent's strategy. Furthermore, as Shoham et al. have stated, many regret-minimizing algorithms can also guarantee safety in addition to universal consistency.

These benefits can be extended to the case where the agent faces reactive opponents as well. Instead of considering single actions at each time step, our modified regret framework considers multi-period strategies by dividing up the

*E-mail address:* ychang@ISI.EDU.

0004-3702/$ – see front matter 
doi:10.1016/j.artint.2006.12.007

sequence of games into intervals. The regret criterion can now take into account reactive strategies such as "Tit-for-Tat". Two issues arise from this setup: 1) an action's potential reward can no longer be observed unless that action is actually played, since the reward depends on the opponent's current reactive strategy, which cannot be observed, and 2) computational complexity grows exponentially as we consider longer intervals and a larger set of more complex strategies. The first issue can be resolved by using a no-regret algorithm such as Auer, Cesa-Bianchi, Freund, and Schapire's EXP3 algorithm [1], which extends Freund and Schapire's multiplicative weight algorithm [6] to the case of partial information. We propose to alleviate the second issue by choosing the set of possible strategies carefully and by incorporating learning algorithms as experts in a modified version of the EXP3 algorithm [4].

## 2. Mathematical background

For most of this article, we will focus our attention on repeated games, though the techniques described can potentially be extended to stochastic games. The repeated game setting already captures much of the complexity of the multi-agent learning problem, where we need to focus on our ability to learn about and react to the opponent. In a stochastic game, we also need to learn about and adapt to the external environment. Here we will focus on modeling the states of the opponent and set aside the problem of additionally modeling the states of the external environment at the same time.

During each stage game of the repeated game, each player simultaneously chooses to play a particular action $a_i \in A_i$ and receives reward based on the joint action taken. We use the terms *policy* and *strategy* interchangeably. While Nash equilibrium is the accepted solution concept for a single stage game, in repeated game and stochastic game settings, modern game theory often takes a more general view of optimality, a view that has also gained acceptance in the machine learning community [3,5]. The key difference is the treatment of the history of actions taken in the game. Here we define a behavioral strategy $\beta : H \to A_i$, where $H = \bigcup_t H^t$ and $H^t$ is the set of all possible histories of length $t$. Histories are observations of joint actions, $h^t = (a_i, a_{-i}, h^{t-1})$. For simplicity, we will assume $A = A_1 = A_2$.

**Definition 1.** A $\tau$-length behavioral strategy $\beta^\tau : H^\tau \to A$ is a mapping from the set of all possible histories $H^\tau$ to actions $a \in A$. Let $B^\tau$ be the set of all possible $\tau$-length behavioral strategies $\beta^\tau$.

We note that $|B^\tau| = |A|^{|A|^{2\tau}}$. In the case where we take $H^t = H$, we could even consider learning algorithms themselves to be a possible "behavioral strategy" for playing a repeated game.

This definition of our strategy space is clearly more powerful, and allows us to define a much larger set of potential equilibria. However, when the opponent is not rational, it is no longer advantageous to find and play an equilibrium strategy. In fact, given an arbitrary opponent, the Nash equilibrium strategy may return a lower payoff than some other action. Indeed, the payoff may be worse than the original Nash equilibrium value. Thus, we turn to regret minimization algorithms.

### 2.1. Regret-minimization

In repeated games, the standard regret minimization framework enables us to perform almost as well as the best action, if that single best action were played in every time period. We will frequently refer to the EXP3 algorithm (and its variants) explored by Auer et al. [1] as an example of this type of algorithm. In the original formulation of EXP3, we choose single actions to play, but we do not get to observe the rewards we would have received if we had chosen different actions.

We need to be a bit more precise about the definition of regret here. Auer et al. consider an adversarial setting where the reward function is actually a sequence of reward functions that change at each time period. We denote the cumulative reward for executing an algorithm $H$ for $T$ time periods by $R_H$, and this is compared with the cumulative reward the agent could have received had it chosen to execute a fixed action $a$ for all $T$ time periods, $R_{\max} = \max_a R_a$. Since the algorithm $H$ is randomized, we will be discussing *expected regret* $R_{\max} - \mathbf{E}[R_H]$. The authors show that the performance of EXP3 exhibits an expected regret bound of $2\sqrt{e-1}\sqrt{TK\ln K}$, where $K$ is the number of action choices, and $T$ is the number of rounds we play the game. In situations where the rewards for all possible actions are observed at each period, this upper bound on the expected regret can be reduced to $O(\sqrt{T\ln K})$.

Generally speaking, these regret-minimizing algorithms hedge between possible actions by keeping a weight for each action that is updated according to the action's historical performance. The probability of playing an action is then its fraction of the total weights mixed with the uniform distribution. Intuitively, better experts perform better, get assigned higher weight, and are played more often. Sometimes these algorithms are called experts algorithms, since we can think of the actions as being recommended by a set of experts. This set is also referred to as our *comparison class*. This comparison class provides a clear means of evaluating our algorithm's performance by pegging this evaluation metric on a set of strategies and assumptions that we know how to execute.

It is important to note that most of these existing methods only compare our performance against strategies that are best responses to what are often called *oblivious* or *myopic* opponents. That is, the opponent does not learn or react to our actions, instead playing a pre-selected, but possibly arbitrary, fixed string of actions. Under most circumstances, however, we might expect an intelligent opponent to change their strategy as they observe our own sequence of plays.

For example, consider the game of repeated Prisoner's Dilemma. If we follow the oblivious opponent assumption, then the best choice of action on hindsight would always be to defect, since we're assuming that the oblivious opponent will not change his next action in response to our defection. This approach would assign low scores (high regret) to cooperative strategies, and thus miss out on the chance to earn higher rewards by cooperating with opponents such as a Tit-for-Tat opponent, which cooperates with us as long as we also cooperate. These opponents can be called *reactive* opponents.

Our extension of the regret framework deals with reactive opponents by expanding our comparison class of strategies to include reactive, or behavioral, strategies. Now, instead of only comparing against the best action from the stage game assuming that the opponent's action stays fixed, we can also compare our performance against strategies such as "Tit-for-Tat" with the assumption that the opponent chooses its actions in response to our strategy. To make this precise, we divide the sequence of games into *phases* of length $\lambda$, and within each phase, we play the same behavioral strategy for $\lambda$ stage games. Thus, instead of choosing actions from $A$ during each stage game, we are instead choosing a behavioral strategy from $B^\lambda$ to play during each phase. Now, if we choose the strategy "Tit-for-Tat", we will be able to observe that it produces higher cumulative rewards over $\lambda$ stage games than a "Always Defect" strategy over the same length phase.

In related work, Mannor and Shimkin [8] propose a similar "super-game" framework for extending the regret-minimization framework to handle stochastic games. In this super-game framework, the phases allow the agent to observe the external environment's reaction to its strategy. Our basic extensions to EXP3 follow the same lines, but deal specifically with modeling a reactive opponent rather than a Markov external environment. We then propose further extensions to deal with the overwhelming computational complexity of this setup. Mannor and Shimkin focus on an alternate method based on the empirical Bayes envelope to provide a more efficient algorithm with slightly weaker guarantees than in their super-game setup. de Farias and Meggido [5] also explore the problem of optimizing behavior against a reactive opponent, but they use a different performance metric rather than regret.

## 3. Extending the experts framework

We now describe in detail the basic extensions to the experts framework that allow us to capture the added power of behavioral strategies. Instead of choosing actions from $A$, we choose behavioral strategies from $B^\tau$. $B^\tau$ also replaces $A$ as our comparison class, essentially forcing us to compare our performance against more complex and possibly better performing strategies. While executing $\beta^\tau \in B^\tau$ for a phase consisting of $\lambda$ stage games, the agent receives reward at each time step, but does not observe the rewards it would have received had it played any of its other possible strategies. This is reasonable since the opponent may adapt differently as a particular strategy is played, causing a different cumulative outcome over $\lambda$ time periods. Thus, the opponent could be an arbitrary black-box opponent or perhaps a fixed finite automaton.

For example, we might consider an opponent whose action choices only depend on the previous $\tau$-length history of joint actions. Thus, we can construct a Markov model $M$ of our opponent using the set of all possible $\tau$-length histories as the state space. If the optimal policy $\beta$ in this MDP is ergodic, we can use the $\epsilon$-return mixing time $\nu_{\beta,M}$ [7] of the induced Markov chain as our choice of $\lambda$, since this would give us a good idea of the average rewards possible with this policy in the long run. We will usually assume that we are given $\nu = \max_{\beta \in B^\tau} \nu_{\beta,M}$, or we simply choose a sufficiently large fixed $\lambda$.

Thus, if we are executing a policy $\beta$ learned on a particular opponent model $M$, then we must run the policy for at least $\nu_{\beta,M}$ time periods to properly estimate the benefit of using that policy. Setting an expert's commitment time horizon $\lambda = \nu$ ensures that the expert receives good estimates of a policy's value during each $\lambda$-length phase, which we denote by its total reward during that phase, $r_\beta(t, t+\lambda-1)$. Over $T$ time periods, the cumulative reward is then $R_\beta = \sum_{i=1}^{\lfloor T/\lambda \rfloor} r_\beta((i-1)\lambda+1, i\lambda)$.

Alternatively, we may be given a fixed phase length $\lambda$, and we would like to be able to evaluate all possible strategies in order to choose the optimal strategy. This would entail enumerating all possible behavioral strategies over $\lambda$ periods. A hedging algorithm $H$ that switches between this set of $B^\lambda$ strategies and picks a $\beta_i$ at each phase $i$ would get a cumulative reward denoted

$$R_H = \sum_{i=1}^{\lfloor T/\lambda \rfloor} r_{\beta_i}\big((i-1)\lambda+1, i\lambda\big).$$

The expected regret of following algorithm $H$ is thus $\max_{\beta\in B^\lambda} R_\beta - \mathbf{E}[R_H]$. However, there are still $|A|^{|A|^{2\lambda}}$ possible strategies to evaluate. Not only would this take a long time to try each possible strategy, but the regret bounds also become exceedingly weak. The expected regret after $T$ time periods is:

$$2\sqrt{e-1}|A|^{\lambda+|A|^{2\lambda}/2}\sqrt{T\lambda\ln|A|}.$$

Clearly this amounts to a computationally infeasible approach to this problem. In traditional MDP solution techniques, we are saved by the Markov property of the state space, which reduces the number of strategies we need to evaluate by allowing us to re-use information learned at each state. Without any assumptions about the opponent's behavior, as in the classic regret minimization framework, we cannot get such benefits.

## 4. Learning algorithms as experts

However, we might imagine that not all policies are useful or fruitful ones to explore, given our choice of a fixed commitment length of $\lambda$. In fact, in most cases, we probably have some rough idea about the types of opponent models that may be appropriate for a given domain. For example, in our Prisoner's Dilemma example, we might expect that our opponent is either a Tit-for-Tat player, an Always-Defect or Always-Cooperate player.

Given particular assumptions about possible opponent models, we may then be able to use a learning algorithm to estimate the model parameters based on observed history. These learning algorithms can be viewed as experts that recommend a particular behavioral strategy $\beta$ to be played during each game phase. The behavioral strategies discussed above are simply experts that recommend the same strategy $\beta_i$ to be played during every phase. The traditional static experts would recommend that the same action be played at every stage game during every game phase.

**Definition 2.** Given a set of experts $E$, the regret we obtain by following an algorithm $H$ is defined as $\max_{e\in E} R_e - \mathbf{E}[R_H]$, where each expert $e$ outputs a recommended behavioral strategy $\beta \in B^\lambda$ at each phase of the repeated game.

For example, we might use a learning expert if we believe that the opponent may be Markov in the $\tau$-length history of joint actions, as discussed in the previous section. We can construct a Markov model of the opponent and use an efficient learning algorithm (such as E3 from Kearns and Singh [7]) to learn the $\epsilon$-optimal policy in time polynomial in the number of states, $|A|^{2\tau}$. We would be able to efficiently learn the best response strategy to opponents such as the one shown in Fig. 1, in this case as long as we choose $\tau \geqslant 4$. In contrast, the hedging algorithm needs to evaluate each of the exponentially large number of possible policies, namely $|A|^{|A|^{2\tau}}$ possible policies.

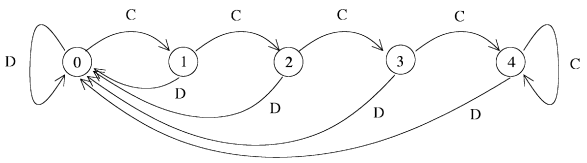


Fig. 1. A possible opponent model with five states. Each state corresponds to the number of consecutive "Cooperate" (C) actions we have just played. "D" stands for the "Defect".

Of course, by using a small number of learning experts rather than the entire set of behavioral strategies, we can no longer guarantee regret minimization over all possible policies. As we will discuss in the following section, we can choose a subset of fixed policies against which we can compare the performance of any learning algorithms we decide to use, and we can guarantee no-regret relative to this subset of fixed policies, as well as relative to the recommended strings of actions produced by the learning algorithms.

In some ways, using learning algorithms as experts simply off-loads the exploration from the experts framework to each individual learning algorithm. The computational savings occur because each learning algorithm makes particular assumptions about the structure of the world and of the opponent, thus enabling each expert to learn more efficiently than hedging between all possible strategies.

## 5. The hedged learner

Since the chosen learning algorithms will sometimes fail to output good policies, we propose to incorporate them as experts inside a hedging algorithm that hedges between a set of experts that includes the learners. This allows the hedging algorithm to switch to using the other experts if a particular learning algorithm fails. It might fail due to incorrect opponent assumptions, such as in the previous section's example if we chose $\tau < 4$, or the learning algorithm may simply be ill-suited for the particular domain.

Here we outline one method for adding learning experts into a regret-minimization algorithm such as Auer et al.'s EXP3 alongside other static experts. It is straightforward to extend these results to other variants of EXP3 such as EXP3.P.1, which guarantees similar bounds that hold uniformly over time and with probability one. We are given $K$ *static experts*, each of which plays a single pure action $a$ at every stage game. We need to add $L$ reactive experts, which may either be learning algorithms or behavioral strategies. For evaluation, the $K$ static experts have $\nu_k = 1, k \in K$, since they assume that the opponent's action string is independent of our agent's action choices. For all $l \in L$, we assume we are given $\nu_l > 1$ based on the policies these experts can produce and their assumptions regarding the opponent model. Since this may not be the true mixing time, we will often call these $\nu_l$ the *commitment length* required by expert $l$. When it is clear from context, we will often write $K$ and $L$ as the number of experts in the sets $K$ and $L$, respectively.

*Hierarchical hedging*: Let $H_0$ denote the top-level hedging algorithm. Construct a second-level hedging algorithm $H_1$ composed of all the original $K$ static strategies. Use $H_1$ and the learning algorithms as the $L + 1$ experts that $H_0$ hedges between.

In contrast, a naive approach for adding learning experts might be to run each of the $K + L$ experts for the same amount of time, i.e. the longest commitment phase required by any of the static experts or learning algorithms, $\lambda_{\max} = \max_{i \in K \cup L} \nu_i$. This naive approach suffers from two main drawbacks, both stemming from the same issue. Because the naive approach follows all experts for $\lambda_{\max}$ periods, it follows the static experts for longer than necessary. Intuitively, this slows down the algorithm's adaptation rate. Furthermore, we also lose out on much of the safety benefit that comes from hedging between the pure actions. Whereas a hedging algorithm over the set of pure action static experts with $\lambda = 1$ is able to guarantee that we attain at least the safety (minimax) value of the stage game, this is no longer true with the naive approach and $\lambda > 1$ since we have not included all possible $\lambda_{\max}$-length behavioral experts. This removes uncertainty regarding the next action, and thus many opponent strategies would be able to exploit this determinism. For example, consider an opponent that runs a behavioral strategy that simply plays the best response to the last period's observed action. Each of the $K$ static experts may then incur high loss when it is run for $\lambda_{\max}$ periods. Hierarchical Hedging addresses these issues.

**Proposition 3.** *Suppose we have a set $K$ of static experts, and a set $L$ of reactive experts with commitment lengths $\lambda_i$, $\max_i \lambda_i > |K|$. We can devise an algorithm with regret bound*:

$$2\sqrt{e-1}\left(\sqrt{TK\ln K} + \sqrt{\lambda_{\max} T(L+1)\ln(L+1)}\right).$$

**Proposition 4.** *The Hierarchical Hedging algorithm (HH) will attain an asymptotic average reward at least close to the safety value, or minimax value $v$, of the stage game: For any $\epsilon > 0$, $\liminf_{T\to\infty} R_{HH}/T - v \geqslant -\epsilon$ almost surely.*

Hierarchical hedging thus provides one method for devising computationally tractable regret-minimizing algorithms that are able to perform well against a large set reactive opponents, achieve no-regret against arbitrary opponents, and guarantee safety.

## 6. Issues to overcome and conclusion

As satisfying as the no-regret framework may seem it be, it isn't a panacea for the complex problems we need to deal with in multi-agent learning problems. As opponent strategies grow more complex, our recognition process and possible strategic responses grow more complex as well. Computational time grows accordingly. We will need to pursue and develop further techniques to reduce this computational load. Work in this direction includes constructing efficient learners that depend only on the VC dimension of the concept class rather than the number of possible experts in this class [9]. We can also try to initially use small opponent models and only enlarge these models using state-splitting techniques when it becomes necessary [2].

However, even given these challenges, one benefit of the regret-minimization approach is that it evaluates our performance relative to a comparison class of strategies that we know we can execute. Thus, given some fixed computational constraints, we can use no-regret approaches to achieve agent performance that is comparable (or better than) the best strategy among a set of strategies that we know we can compute within our constraints. This capability holds no matter what type of opponent the agent ends up facing in the game.

We've seen that regret-minimization approaches provide us with a number of benefits when applied to the multi-agent learning problem. Specifically, these techniques are likely to be most useful in the fifth area of research that Shoham et al. describe, namely prescriptive, non-cooperative problems. These techniques allow us to create algorithms that guarantee safety and exhibit no-regret behavior against arbitrary opponents. Moreover, the extensions we've described in this article allow the inclusion of behavioral strategies and even learning algorithms in our comparison class, thus ensuring that our agent performs well against not only static opponents, but also reactive opponents as well. No-regret algorithms can be said to only perform as well as the comparison class by which they are measured. By opening the door to these larger comparison classes, we can now develop regret-minimizing algorithms that not only exhibit no regret but also truly perform well in a wide variety of settings.

## References

[1] P. Auer, N. Cesa-Bianchi, Y. Freund, R.E. Schapire, Gambling in a rigged casino: the adversarial multi-armed bandit problem, in: Proceedings of the 36th Symposium on Foundations of Computer Science, 1995.
[2] Y. Chang, P.R. Cohen, C.T. Morrison, W. Kerr, R.S. Amant, The Jean system, in: Proceedings of the International Conference on Development and Learning, 2006.
[3] Y. Chang, L.P. Kaelbling, Playing is believing: The role of beliefs in multi-agent learning, in: Advances in Neural Information Processing Systems, 2001.
[4] Y. Chang, L.P. Kaelbling, Hedged learning: Regret-minimization with learning experts, in: International Conference on Machine Learning, 2005.
[5] D.P. de Farias, N. Meggido, How to combine expert (or novice) advice when actions impact the environment, in: Advances in Neural Information Processing Systems, 2004.
[6] Y. Freund, R.E. Schapire, Adaptive game playing using multiplicative weights, Games and Economic Behavior 29 (1999) 79–103.
[7] M. Kearns, S. Singh, Near-optimal reinforcement learning in polynomial time, in: Proceedings of the International Conference on Machine Learning, 1998.
[8] S. Mannor, N. Shimkin, The empirical Bayes envelope approach to regret minimization, Technion Technical Report EE-No1261, 2000.
[9] A. Strehl, C. Mesterharm, M.L. Littman, H. Hirsh, Experience-efficient learning in associative bandit problems, in: Proceedings of the International Conference on Machine Learning, 2006.